

IE 410 – INTRODUCTION TO ROBOTICS

Lab-6 report

**Controlling Turtlesim Node using Python code,
implementing P controller and implementing PID controller.**

Team M410:

201901011 – HIMANSHU DUDHATRA

201901024 – DHAVALSINH RAJ

201901100 – SHUBHAM PATEL

201901145 – GARGEY PATEL

- **Creating Catkin package**

First go to the 'src' folder of catkin workspace that we have created.

```
$ cd catkin_ws/src
```

Now create a catkin package named linuxsquad_pkgforlab4 by using the command given below.

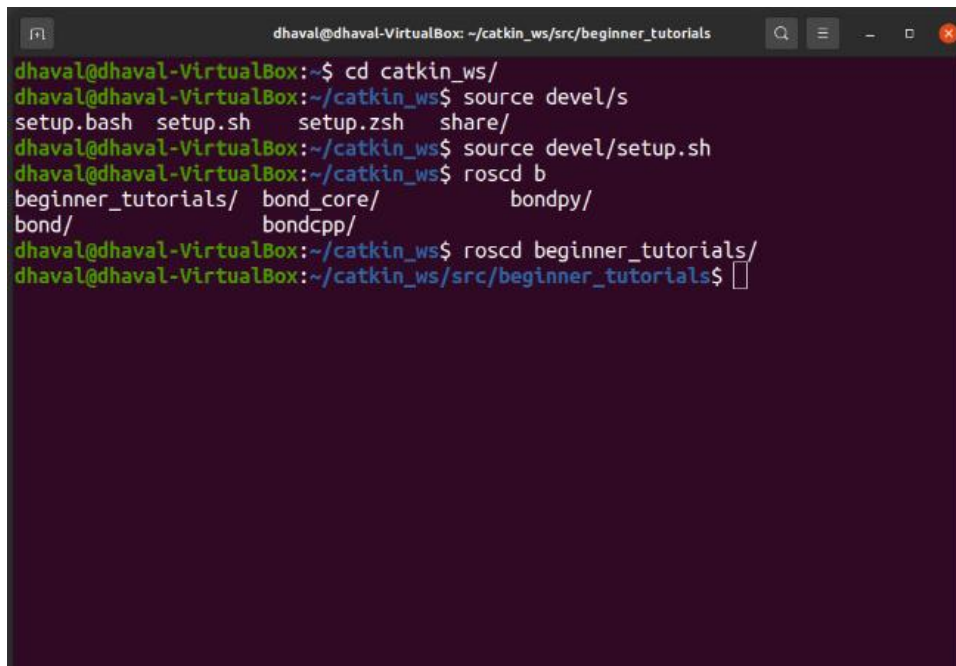
```
$ catkin_create_pkg beginner_tuts std_msgs rospy roscpp
```

Now we will execute catkin_make.

```
$ catkin_make
```

Now we need to source the generated setup file.

```
$ source devel/setup.bash
```

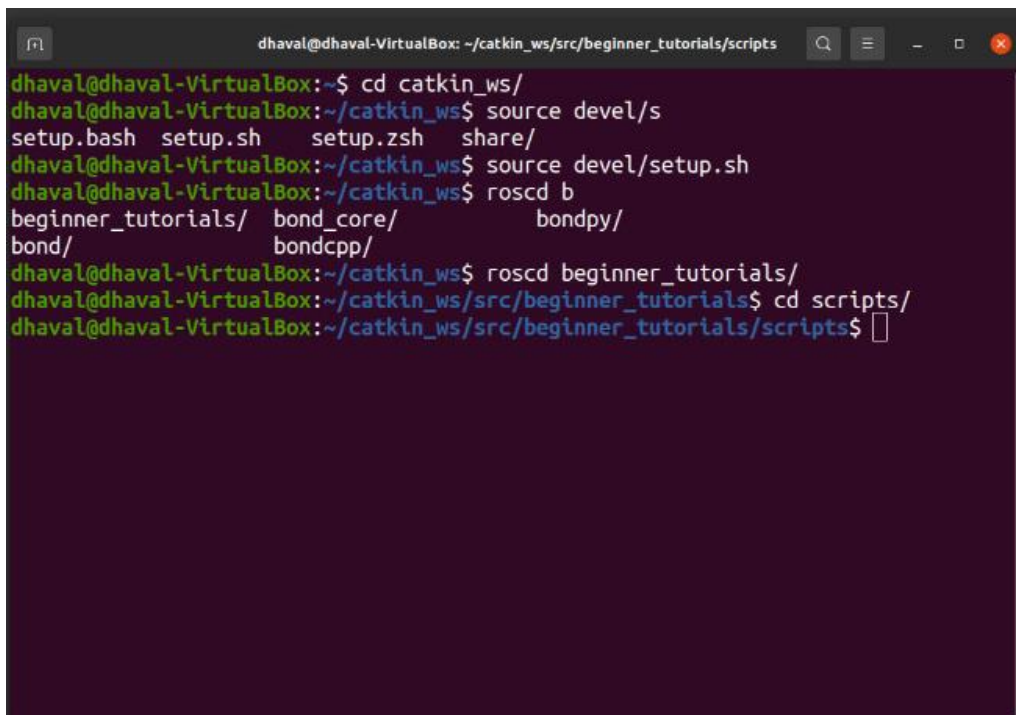
A terminal window titled 'dhaval@dhaval-VirtualBox: ~/catkin_ws/src/beginner_tutorials' showing the following commands and output:

```
dhaval@dhaval-VirtualBox:~$ cd catkin_ws/  
dhaval@dhaval-VirtualBox:~/catkin_ws$ source devel/s  
setup.bash setup.sh setup.zsh share/  
dhaval@dhaval-VirtualBox:~/catkin_ws$ source devel/setup.sh  
dhaval@dhaval-VirtualBox:~/catkin_ws$ roscd b  
beginner_tutorials/ bond_core/ bondpy/  
bond/ bondcpp/  
dhaval@dhaval-VirtualBox:~/catkin_ws$ roscd beginner_tutorials/  
dhaval@dhaval-VirtualBox:~/catkin_ws/src/beginner_tutorials$
```

- **Turtlesim_move.py and Turtlesim_cleaner.py**

Now, we will change the directory to linuxsquad_pkglab6 and then will create a `scripts` directory in the linuxsquad_pkglab6 directory. Follow these commands.

```
$ roscd linuxsquad_pkgforlab6
$ mkdir scripts
$ cd scripts
```



```
dhaval@dhaval-VirtualBox: ~/catkin_ws/src/beginner_tutorials/scripts
dhaval@dhaval-VirtualBox:~$ cd catkin_ws/
dhaval@dhaval-VirtualBox:~/catkin_ws$ source devel/s
setup.bash setup.sh setup.zsh share/
dhaval@dhaval-VirtualBox:~/catkin_ws$ source devel/setup.sh
dhaval@dhaval-VirtualBox:~/catkin_ws$ roscd b
beginner_tutorials/ bond_core/ bondpy/
bond/ bondcpp/
dhaval@dhaval-VirtualBox:~/catkin_ws$ roscd beginner_tutorials/
dhaval@dhaval-VirtualBox:~/catkin_ws/src/beginner_tutorials$ cd scripts/
dhaval@dhaval-VirtualBox:~/catkin_ws/src/beginner_tutorials/scripts$
```

- **Turtlesim_cleaner.py**

```
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
from turtlesim.msg import Pose
import math
import time
from std_srvs.srv import Empty

x=0
y=0
yaw=0

def poseCallback(pose_message):
    global x
    global y, yaw
```

```

x= pose_message.x
y= pose_message.y
yaw = pose_message.theta

#print "pose callback"
#print ('x = {}'.format(pose_message.x)) #new in python 3
#print ('y = %f' %pose_message.y) #used in python 2
#print ('yaw = {}'.format(pose_message.theta)) #new in python 3

def move(speed, distance, is_forward):
    #declare a Twist message to send velocity commands
    velocity_message = Twist()
    #get current location
    global x, y
    x0=x
    y0=y

    if (is_forward):
        velocity_message.linear.x =abs(speed)
    else:
        velocity_message.linear.x =-abs(speed)

    distance_moved = 0.0
    loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10
times a second)
    cmd_vel_topic='/turtle1/cmd_vel'
    velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist,
queue_size=10)

    while True :
        rospy.loginfo("Turtlesim moves forwards")
        velocity_publisher.publish(velocity_message)

        loop_rate.sleep()

        #rospy.Duration(1.0)

        distance_moved = abs(0.5 * math.sqrt(((x-x0) ** 2) + ((y-
y0) ** 2)))

        print (distance_moved);
        if not (distance_moved<distance):
            rospy.loginfo("reached")
            break

    #finally, stop the robot when the distance is moved
    velocity_message.linear.x =0
    velocity_publisher.publish(velocity_message)

def rotate (angular_speed_degree, relative_angle_degree, clockwise):

    global yaw
    velocity_message = Twist()
    velocity_message.linear.x=0
    velocity_message.linear.y=0
    velocity_message.linear.z=0
    velocity_message.angular.x=0

```

```

velocity_message.angular.y=0
velocity_message.angular.z=0

#get current location
theta0=yaw
angular_speed=math.radians(abs(angular_speed_degree))

if (clockwise):
    velocity_message.angular.z =-abs(angular_speed)
else:
    velocity_message.angular.z =abs(angular_speed)

angle_moved = 0.0
loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times
a second)
cmd_vel_topic='/turtle1/cmd_vel'
velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist,
queue_size=10)

t0 = rospy.Time.now().to_sec()

while True :
    rospy.loginfo("Turtlesim rotates")
    velocity_publisher.publish(velocity_message)

    t1 = rospy.Time.now().to_sec()
    current_angle_degree = (t1-t0)*angular_speed_degree
    loop_rate.sleep()

    if (current_angle_degree>relative_angle_degree):
        rospy.loginfo("reached")
        break

#finally, stop the robot when the distance is moved
velocity_message.angular.z =0
velocity_publisher.publish(velocity_message)

def go_to_goal(x_goal, y_goal):
    global x
    global y, yaw

    velocity_message = Twist()
    cmd_vel_topic='/turtle1/cmd_vel'

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

```

```

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)

        #print velocity_message.linear.x
        #print velocity_message.angular.z
        print ('x=', x, 'y=', y)

    if (distance < 0.01):
        break

def setDesiredOrientation(desired_angle_radians):
    relative_angle_radians = desired_angle_radians - yaw
    if relative_angle_radians < 0:
        clockwise = 1
    else:
        clockwise = 0
    print (relative_angle_radians)
    print( desired_angle_radians)
    rotate(30 ,math.degrees(abs(relative_angle_radians)), clockwise)

def gridClean():

    desired_pose = Pose()
    desired_pose.x = 1
    desired_pose.y = 1
    desired_pose.theta = 0

    moveGoal(desired_pose, 0.01)

    setDesiredOrientation(degrees2radians(desired_pose.theta))

    move(2.0, 9.0, True)
    rotate(degrees2radians(20), degrees2radians(90), False)
    move(2.0, 9.0, True)
    rotate(degrees2radians(20), degrees2radians(90), False)
    move(2.0, 1.0, True)
    rotate(degrees2radians(20), degrees2radians(90), False)
    move(2.0, 9.0, True)
    rotate(degrees2radians(30), degrees2radians(90), True)
    move(2.0, 1.0, True)
    rotate(degrees2radians(30), degrees2radians(90), True)
    move(2.0, 9.0, True)
    pass

def spiralClean():
    vel_msg = Twist()
    loop_rate = rospy.Rate(1)
    wk = 4
    rk = 0

    while((currentTurtlesimPose.x<10.5) and (currentTurtlesimPose.y<10.5)):
        rk=rk+1

```

```

        vel_msg.linear.x =rk
        vel_msg.linear.y =0
        vel_msg.linear.z =0
        vel_msg.angular.x = 0
        vel_msg.angular.y = 0
        vel_msg.angular.z =wk
        velocity_publisher.publish(vel_msg)
        loop_rate.sleep()

    vel_msg.linear.x = 0
    vel_msg.angular.z = 0
    velocity_publisher.publish(vel_msg)

if __name__ == '__main__':
    try:

        rospy.init_node('turtlesim_motion_pose', anonymous=True)

        #declare velocity publisher
        cmd_vel_topic='/turtle1/cmd_vel'
        velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist,
        queue_size=10)

        position_topic = "/turtle1/pose"
        pose_subscriber = rospy.Subscriber(position_topic, Pose,
        poseCallback)
        time.sleep(2)

        #move(1.0, 2.0, False)
        #rotate(30, 90, True)
        go_to_goal(1.0, 1.0)
        #setDesiredOrientation(math.radians(90))

    except rospy.ROSInterruptException:
        rospy.loginfo("node terminated.")

```

• Turtlesim_move.py

```

#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist
import math
import time
from std_srvs.srv import Empty

def move(speed, distance, is_forward):
    #declare a Twist message to send velocity commands
    velocity_message = Twist()
    #get current location

```

```

    if (speed > 0.4):
        print('speed must be lower than 0.4')
        return

    if (is_forward):
        velocity_message.linear.x =abs(speed)
    else:
        velocity_message.linear.x =-abs(speed)

    distance_moved = 0.0
    loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10
times a second)
    cmd_vel_topic='/cmd_vel_mux/input/teleop'
    velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist,
queue_size=10)

    t0 = rospy.Time.now().to_sec()

    while True :
        rospy.loginfo("Turtlesim moves forwards")
        velocity_publisher.publish(velocity_message)

        loop_rate.sleep()
        t1 = rospy.Time.now().to_sec()
        #rospy.Duration(1.0)

        distance_moved = (t1-t0) * speed
        print(distance_moved)
        if not (distance_moved<distance):
            rospy.loginfo("reached")
            break

        #finally, stop the robot when the distance is moved
        velocity_message.linear.x =0
        velocity_publisher.publish(velocity_message)

def rotate (angular_speed_degree, relative_angle_degree, clockwise):

    velocity_message = Twist()
    velocity_message.linear.x=0
    velocity_message.linear.y=0
    velocity_message.linear.z=0
    velocity_message.angular.x=0
    velocity_message.angular.y=0
    velocity_message.angular.z=0

    angular_speed=math.radians(abs(angular_speed_degree))

    if (clockwise):
        velocity_message.angular.z =-abs(angular_speed)
    else:
        velocity_message.angular.z =abs(angular_speed)

    angle_moved = 0.0

```



```

    loop_rate = rospy.Rate(10) # we publish the velocity at 10 Hz (10 times
a second)
    cmd_vel_topic='/cmd_vel_mux/input/teleop'
    velocity_publisher = rospy.Publisher(cmd_vel_topic, Twist,
queue_size=10)

    t0 = rospy.Time.now().to_sec()

    while True :
        rospy.loginfo("Turtlesim rotates")
        velocity_publisher.publish(velocity_message)

        t1 = rospy.Time.now().to_sec()
        current_angle_degree = (t1-t0)*angular_speed_degree
        loop_rate.sleep()

        print('current_angle_degree: ',current_angle_degree)

        if (current_angle_degree>relative_angle_degree):
            rospy.loginfo("reached")
            break

    #finally, stop the robot when the distance is moved
    velocity_message.angular.z =0
    velocity_publisher.publish(velocity_message)

def go_to_goal(x_goal, y_goal):
    global x
    global y, z, yaw

    velocity_message = Twist()
    cmd_vel_topic='/turtle1/cmd_vel'

    while (True):
        K_linear = 0.5
        distance = abs(math.sqrt(((x_goal-x) ** 2) + ((y_goal-y) ** 2)))

        linear_speed = distance * K_linear

        K_angular = 4.0
        desired_angle_goal = math.atan2(y_goal-y, x_goal-x)
        angular_speed = (desired_angle_goal-yaw)*K_angular

        velocity_message.linear.x = linear_speed
        velocity_message.angular.z = angular_speed

        velocity_publisher.publish(velocity_message)
        print('x=', x, 'y=',y)

        if (distance <0.01):
            break

if __name__ == '__main__':

```

The screenshot displays a ROS2 environment with a terminal window and a visualization window.

Terminal Window:

```
dhaval@dhaval-VirtualBox: ~/catkin_ws/src/beginner_tutorials$ roslaunch turtlebot3_fake turtlebot3_fake.launch
[INFO] [1647452889.632524313]: Starting turte.py
[INFO] [1647452889.639991360]: Spawning turt
[INFO] [1647453218.275192]: TurtleSim rotates
current_angle_degree: 0.1391744613647461
[INFO] [1647453218.376073]: TurtleSim rotates
current_angle_degree: 9.265036582946777
[INFO] [1647453218.475764]: TurtleSim rotates
current_angle_degree: 18.242368698120117
[INFO] [1647453218.574347]: TurtleSim rotates
current_angle_degree: 27.12350606918335
[INFO] [1647453218.674295]: TurtleSim rotates
current_angle_degree: 36.122918128967285
[INFO] [1647453218.774553]: TurtleSim rotates
current_angle_degree: 45.12329578399658
[INFO] [1647453218.874388]: TurtleSim rotates
current_angle_degree: 54.151504039764404
[INFO] [1647453218.973886]: TurtleSim rotates
current_angle_degree: 63.062682151794434
[INFO] [1647453219.077593]: TurtleSim rotates
current_angle_degree: 72.41595268249512
[INFO] [1647453219.175402]: TurtleSim rotates
current_angle_degree: 81.2069034576416
[INFO] [1647453219.273944]: TurtleSim rotates
current_angle_degree: 90.09625911712646
```

Visualization Window:

The visualization window shows a 2D environment with a blue background. A red turtle icon is visible in the bottom-left corner, representing the simulated robot. A white line indicates the path of the turtle as it moves across the environment.